GAME-KIT SOFTWARE

1. every hardware interface part has its own source + header file

   - advantage:
     * easy to convert it to nearly every plattform (- independent)
     * the low level hardware drivers can be reused in future projects
     * the font and animation part can be reused in future projects
     * the game can be modified easiely without knowing the actual hardware design of this KIT, very convenient and ideal for beginners

2. the main game application is scalable

   e.g.: - the screen resolution is not fixed.
   - the stone size is scalable.
   - the font and animations are user editable.

   - advantage:
     - not fixed to the GAME-KIT specification

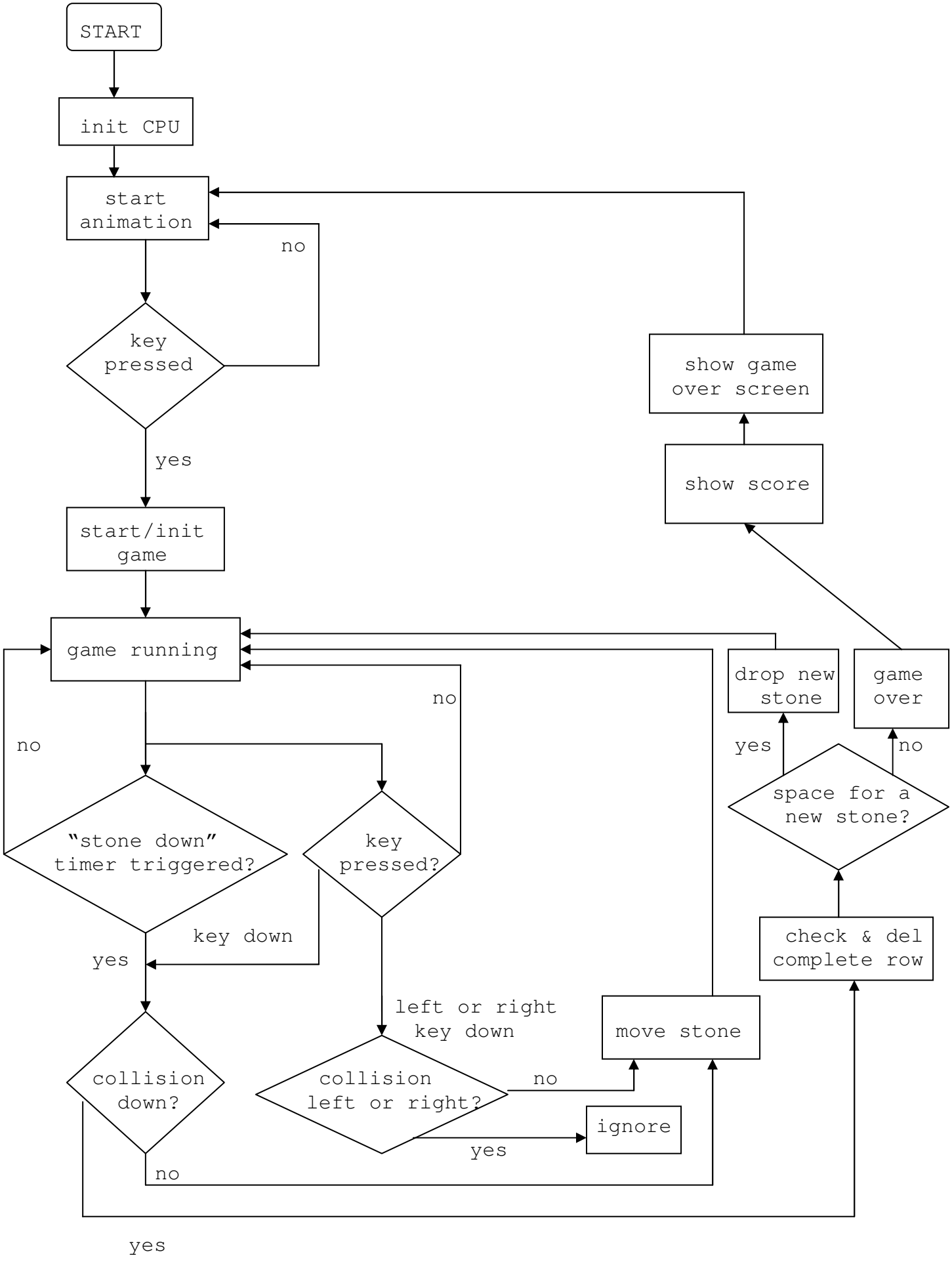
   - disadvantage:
     - not as short/optimized as it could be
     - somehow more complex to develop than an not scalable version

All in all we tried to design an easy to use and useful KIT for many application types such as monitoring and displaying.
Beginner who want to get used to human interfaces, such as keyboard (input) and a relatively big screen (14*20) (output) and want to play around with micro controllers without an enormous effort to learn the schematic design, etc.

I. The sequence of the running program:

1. initialisation of the hardware (internal and external) and variables/registers after power up
   1.1. I/Os
   1.2. timer
   1.3. LED matrix processor
   1.4. variables

2. endless loop consists of:
   2.1. the introduction animation
   2.2. game loop containing:
   - generation of the displayed playfield (gameplay.c + gameplay.h)
   - check whether a collision would occure/occured and if, a check if the user lost the game or still able to play another brick/stone.
   - If game is over: a "game over animation" will be showed, and after a short while back to a. (gameplay.c + gameplay.h)
   - polling of keyboard and processing this data (XX_keyboard.c + XX_keyboard.h)
   - transferring the video memory to display controller processor (XX_ledmatrix.h + XX_ledmatrix.h)
   - random number generation for the next brick/stone
   - etc. (playing sounds/melodies) (XX_sound.h + XX_sound.c)
   - game over animation, (then back to 2.1.) (native_pictures.h + native_pictures.c)


START
init CPU
start animation
no
key pressed
yes
start/init game
game running
no
"stone down" timer triggered?
key down
yes
collision down?
no
yes
key pressed?
no
left or right key down
collision left or right?
no
yes
ignore
move stone
drop new stone
yes
space for a new stone?
no
game over
check & del complete row
show score
show game over screen

II. The structure of the source files:

1. Used source files:

platform independent:

main.c	←	main body of the game application
gameplay.h gameplay.c	←	main functions/procedures of the game (generation of the play field, collision, the movement of the brick/stones (I.2.2.1 - I.2.2.2)
stones.h stones.c	←	stones/bricks declaration + definition header
animation.h animation.c animationdata.h animationdata.c	←	source for playing animations (the animation data itself)
font.h font.c fontdata.h fontdata.c	←	source for displaying custom fonts + text scroller (the font data itself)

platform dependent:

WIN32:

pc_init.h pc_init.c	←	used for initializing the main software components
pc_timers.h pc_timers.c	←	pseudo timer functions for win32 environment, timing is not accurate, just for simple development, testing
pc_sound.h pc_sound.c	←	pseudo sound functions for win32 environment, no sound will be played under win32, just empty function prototypes + and implementations
pc_ledmatrix.h pc_ledmatrix.c emulate the LED Matrix	←	pseudo LED matrix functions for win32 environment, a console window is used to
pc_keyboard.h pc_keyboard.c used to emulate the hardware keys of the GAME-KIT	←	pseudo keyboard functions for win32 environment, the STDIO (pc-keyboard) is

Fujitsu processor (GAME-KIT):

fu_init.h fu_init.c	←	initialization of CPU registers + variables and ISRs
fu_ledmatrix.h fu_ledmatrix.c	←	low level driver for LED matrix of the GAME-KIT
fu_keyboard.h fu_keyboard.c	←	low level driver for the keyboard
fu_timers.h fu_timers.c	←	low level driver for the TIMER ISRs
fu_sound.h fu_sound.c	←	low level driver for the sound/melody generation
fu_battery.h fu_battery.c	←	low level driver for checking and displaying the current battery status